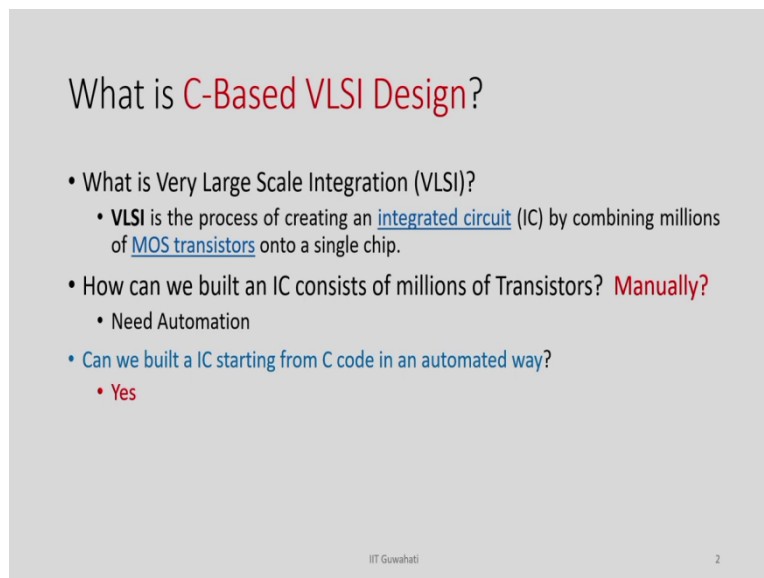


C-Based VLSI Design
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 01
Introduction to C – based VLSI Design
Lecture - 01
Introduction to Course

So, welcome to my course C-Based VLSI design. I am Dr. Chandan Karfa. I am with IIT, Guwahati. I am going to teach this course to you. So, let us try to understand what is C-based VLSI design.

(Refer Slide Time: 00:58)



What is C-Based VLSI Design?

- What is Very Large Scale Integration (VLSI)?
 - VLSI is the process of creating an [integrated circuit](#) (IC) by combining millions of [MOS transistors](#) onto a single chip.
- How can we built an IC consists of millions of Transistors? **Manually?**
 - Need Automation
- Can we built a IC starting from C code in an automated way?
 - **Yes**

IIT Guwahati 2

So, what is VLSI? VLSI is something is the process of creating an integrated circuits by combining million of transistors, right. So, when you try to build a chip or IC with million or billion of transistors that is VLSI and then the question is that how we will build a chip with millions of transistors, right? Is it manually possible to develop such chip? Its answer is no, right.

So, how can you build a chip manually by just doing all things manually is impossible. So, how it is being done? Right. So, how it is done? We need automation. We need some software that will do things for us right. So, we will give some specification and the

automation tool will basically do the whole things for us and will give the integrated circuits finally, right.

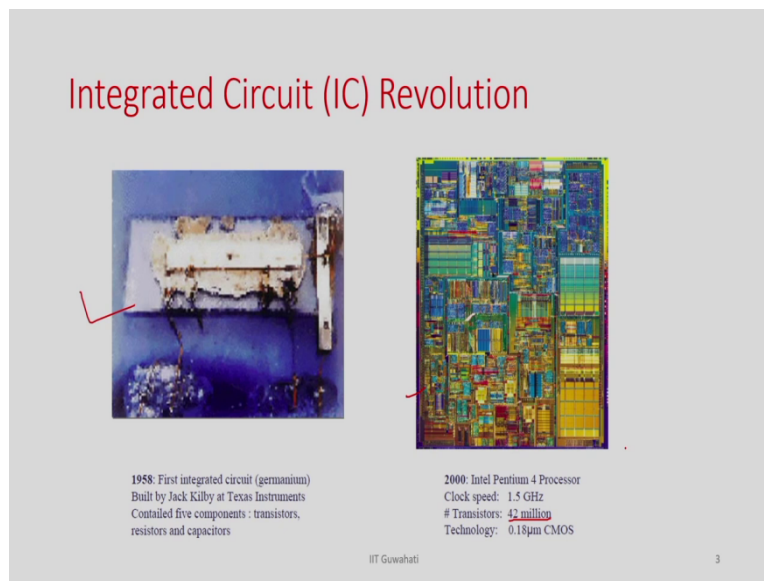
So far we understand that what is IC and how this VLSI chip is built with the help of electronic design automation tools. But, this course asks the question can we start this process from a C code or a high level code? It can be C, C plus plus, python and all. But, can I start this design process from this C?

It is particularly very challenging because you once you think about the hardware. It is basically it has a clock, reset-set, it has some synchronization, synchronous clock - so many things are there. You have to always take care of everything, right. So, it is basically completely different design paradigm, whereas, your C, which is a untimed code.

There is no clock there, right. You just write a code which you just execute top down manner, right. So, top to bottom manner. So, there is no clock involved, there is no adder, multiplier RAM, ROM nothing. So, the question here is that can we develop a software, a automation tool, that will just take a C code and convert to a hardware for me. It is kind of a gateway from a software code to hardware.

So, it is particularly very exciting right. Then in all this trouble, all this complexities of developing a hardware is something we can eliminate. So, that is why this course is so exciting and I am sure that if you go in through this whole courses you will also be very excited and you will learn a lot of things, ok.

(Refer Slide Time: 03:30)




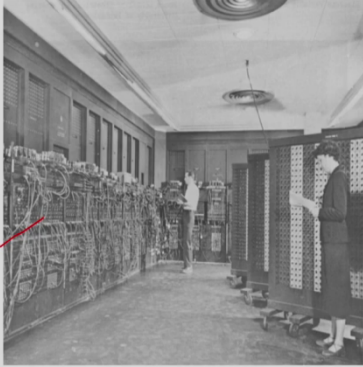
So, in this particular course I just tried to motivate you why this electronic design automation is so important, how this IC revolution comes and how this particular C based VLSI design particularly in the current context is so important and then I am going to conclude this course with the topics to be covered in this course, ok.

So, let us go back to the time 1958; when the first transistor was built. This picture is shown there. So, this was the first transistor built in 1925 in TI. You can understand the size of this transistor. Now, this is 2020. It is a 20 year old picture. It is a Pentium 4 processors, which consists of 42 million number of transistors and the size is you can understand right. So, you can understand this revolution, right.

So, from a tiny single transistor which is something is almost a 2 centimeter long, now we can actually pack millions of transistors into a very small size of die, right.

(Refer Slide Time: 04:38)

How Chips Have Shrunk?



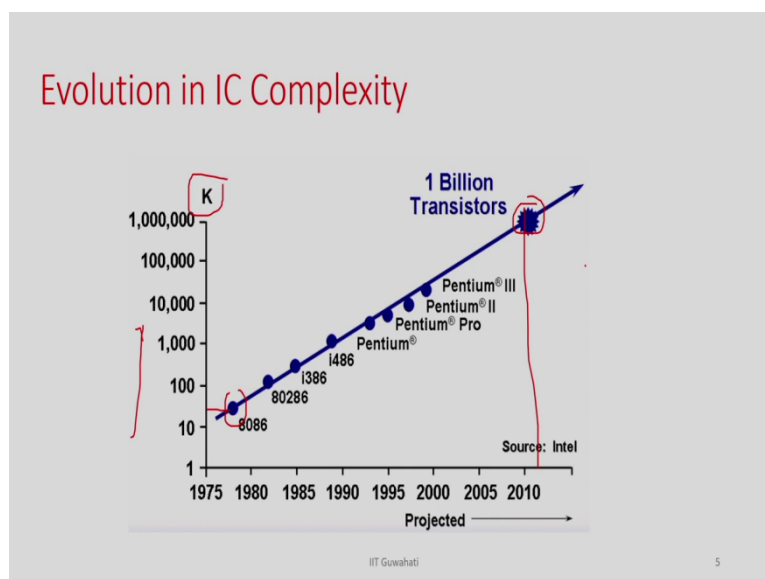
Intel/Altera Stratix 10
~30B transistors

- 1946 in UPenn
- Measured in cubic ft.

IIT Guwahati 4

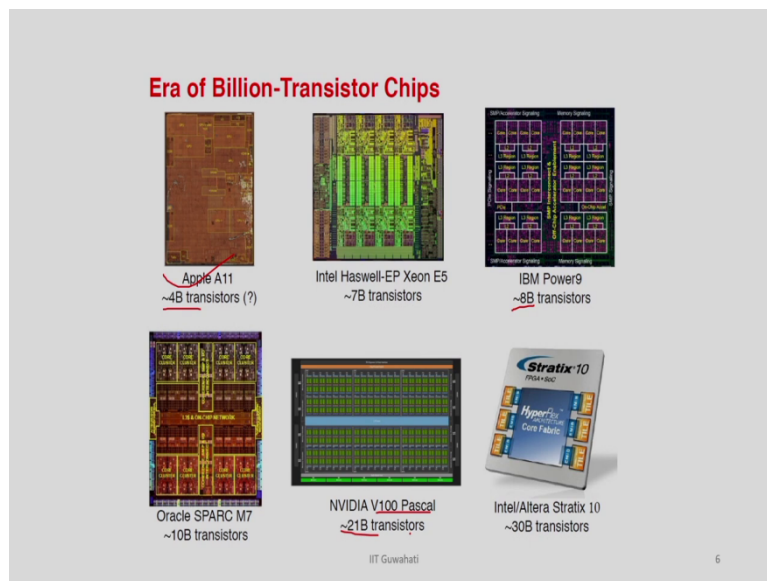
So, I mean there are some other funny pictures. Here this is supposed to be the first computer. You can see this is the size of this computer which is measured in cubic feet, right and now we have this Stratix10 which comes in I think 2018 which has 30 billion transistors. So, you can understand the growth in 50-60 years, the kind of growth that happened in this electronics design industry in this span of 50 or 60 years.

(Refer Slide Time: 05:10)



So, here is the chart that give you this kind of growth chart. So, usually we have the number of transistors in K mentioned in this y axis and this is the year. So, you can see this 8086 have something 15K kind of or say 20K kind of transistors and now, in 2012 and so, we reach a target of 1 billion transistors in a single chip, right. So, this something is just highlight the kind of the growth or the evolution of this IC and whenever you have more things you have more trouble, right. So, the complexity is growing, right.

(Refer Slide Time: 05:54)



So, this is where I just taken certain modern processors say Apple A11. It has 4 billion transistors. This IBM Power9 has around 8 billion transistors and this NVIDIA processor has around 21 billion transistors. So, that you understand the complexity that the current electronic design automation tool has to handle, right.

(Refer Slide Time: 06:19)

How to design with 10B transistors design?

- “This incredible growth rate could not be achieved by hiring an exponentially growing number of design engineers. It was fulfilled by adopting new design methodologies and by **introducing innovative design automation software** at every processor generation”

TABLE 1. INTEL PROCESSORS, 1971-1993.

PROCESSOR	INTRO DATE	PROCESS	TRANSISTORS	FREQUENCY
4004	1971	10 μm	2,300	108 kHz
8080	1974	6 μm	6,000	2 MHz
8086	1978	3 μm	29,000	10 MHz
80286	1982	1.5 μm	134,000	12 MHz
80386	1985	1.5 μm	275,000	16 MHz
Intel 486 DX	1989	1 μm	1.2 M	33 MHz
Pentium	1993	0.8 μm	3.1 M	60 MHz

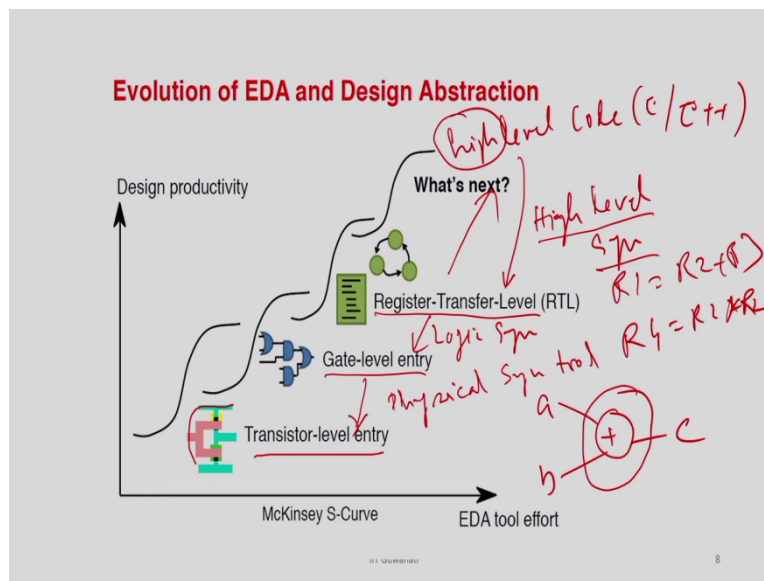
IT Guwahati

7

So, now the question here is how to design this 10 billion transistor design, 20 billion transistor design? Is it something where you hire more electronic engineers. So, they are design engineer, they will build for you. So, you basically hire more people so that you can actually parallelize the task and you do that? No, again it is not, it is impossible or it cannot be achieved.

This growth cannot be achieved by just hiring exponential number of design engineers, right. It is achieved because of this design automation software. Because of this evolution of the electronic design automation software this growth is possible, right.

(Refer Slide Time: 07:04)



So, let us try to understand what is this design automation things I am talking about, right. So, in a chip when you think about a chip or IC it consists of billions of transistors. So, these transistors are nothing but the layout. They are MOS or CMOS or PMOS transistors. So, they are actually layout in physically, right.

So, you basically if you think about the actual circuit, it has billion of transistors, right. So, this is what these transistors. This is actually a physical chip. You cannot replace that. That is how this chip will work, right. So, what is this electronic design automation helps? And this is by due to a design abstraction. So, what is this design abstraction? The abstraction comes that I am not going to look into my chip in the transistor level. I will try to look into a little bit higher abstraction level, right?

What is that? So, I will try to now visualize my circuit at the gate level, right. So, what is gate level? So, instead of transistor PMOS, CMOS, NMOS I am now going to think about my circuit which is a connection of say AND gate, OR gate and so on. So, all these logic gates, right. And what is that? So, one AND gate is something which will abstract probably 5 or 6 transistors.

So, one behavior of 5 or 6 transistor, probably I can combine and I can actually represent that as a AND gate. An AND gate, I know this the behavior, right. So, similarly I can actually

abstract a particular 5 or 6 transistors and I can actually abstract them as a say XOR gate and so on.

So, this is how I can actually reduce the overall complexity by at least 10 x, right. So, this is where the gate level. So, I do not have to think about say 10 billion transistors, rather I can now actually think about 1 billion gate level gates in the circuit of a circuit consists of 1 billion gates. But, how this things will done? This will do by a electronic design automation tool, right.

So, this is what is the physical synthesis tool. So, the EDA software will do this conversion for me. I will visualize my circuit at the gate level and I will complete all these connections, interconnection. All the functionalities will implemented at the gate level which is at least 10x less complex design for me and then I will rely this physical synthesis tool. They will convert those gate level gate level circuit into transistor level circuits, right.

So, that is something is the design abstraction, but still as I mentioned this 10 billion to 1 billion, that is still not possible to handle physically. So, then we should think about more abstraction level, right. So, when you go to one more higher level of abstraction and that is called register transfer level design, what is called RTL.

So, in that register transfer level behavior I am not going to bother about how a adder is implemented, how is a circuit of a multiplier? I will just represent my whole circuit as a transfer of register values, right. That means, at every clock, which particular register getting updated by the value of which registers, right.

So, I am going to write my behavior like $R1 = R2 + R3$ and so on, right. So, say $R4 = R2 * R3$, something like this. So, I do not have to bother about the detail implementation of AND gate, a MUX, DEMUX or say multiplier. What I am going to just write that there is a full adder, right. So, I will draw the circuit like there is a schematic. So, instead of a whole complex circuit, I will just put a and I will just put a, b and c. So, this is C, is nothing but a plus b and so on.

So, this whole design complexity of addition operation or the multiplication operations, those things now getting abstracted out. So, in the RTL level I will get more reduction in the design

complexity. So, it may be say 1 billion to at least again 10x. I mean if not more at least we will get a advantage of 10x size reduction. So, it may be say around 10 million now.

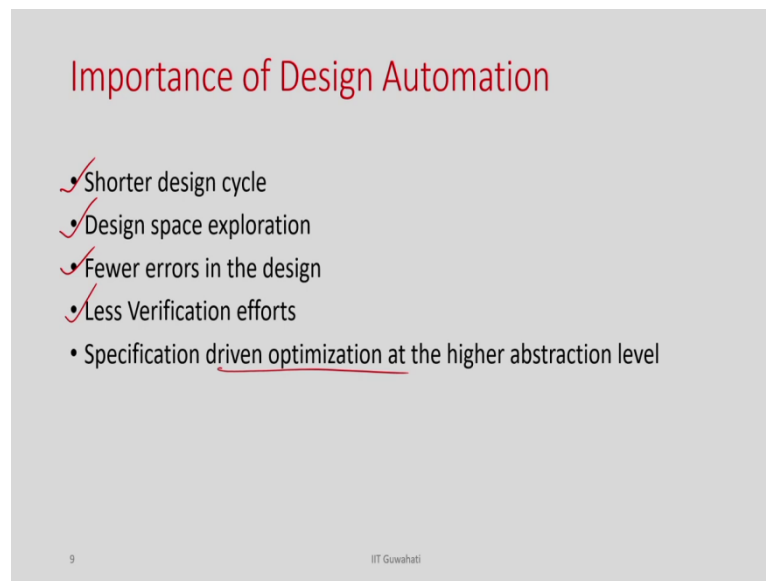
So, that is how this abstraction goes to the register transfer level and again, I have synthesis tool or the EDA tools which is called logic synthesis tool. Logic synthesis tool so, that particular tool actually convert this register transfer level to gate level automatically, right. So, in industry, EDA industry, in the electronics industry people try to design the circuits in RTL. That is the entry point of the circuit because this is the highest level of abstraction, right.

So, you do not are considering the whole gate level thing, this is the highest level thing. But is it so, that as I mentioned the 10 million line of code is still feasible? It is something, sometime getting difficult once the size of the design is getting increased, right. So, again so, that is where we need to go more abstraction level, right. And that is where this particular course comes into picture that let us go to one more abstraction level.

So, you have a high level of code which is C, C plus plus and then you actually use. Again you develop some software EDA tool that will convert this high level code into RTL code and that is what is called this high level synthesis tool. High level synthesis and that is where I talked about this course. This is where this course is particularly poised in. This is how how we actually develop a VLSI design from a high level code written in C or C plus plus, right.

So, that means, we are going to go into detail of this tool. How this particular tool converts this high level code edited in C, C plus into a equivalent register transfer level designs. Now what are the process, what are the things, you have to cover? That particular things we are going to cover in this course in detail, right.

(Refer Slide Time: 13:39)



The slide is titled "Importance of Design Automation" in red text. Below the title is a bulleted list of five points, each preceded by a red checkmark. The points are: "Shorter design cycle", "Design space exploration", "Fewer errors in the design", "Less Verification efforts", and "Specification driven optimization at the higher abstraction level". The word "driven" is underlined in red. At the bottom left of the slide is the number "9", and at the bottom center is "IIT Guwahati".

- ✓ Shorter design cycle
- ✓ Design space exploration
- ✓ Fewer errors in the design
- ✓ Less Verification efforts
- Specification driven optimization at the higher abstraction level

So, let us move on. So, I mean, I hope you probably understand that what are the importance of this design automation? Obviously, the shorter design cycle, right. So, we understand that if we just write a code in higher abstraction level, if the size of the code is less and hence we will need a less time because the tool automatically within minutes or seconds it can convert a high level design into a low level design.

It may, it might take maximum hours not days or months, right. So, whereas if you have to develop certain things in gate level or transistor level that might takes years, right. So, so obviously, if you develop in higher abstraction level, your overall time to develop the whole IC will be less, right. So, you have a shorter design cycle.

So, the biggest advantage is the design space exploration. Say, suppose you want to develop a general purpose processor and then you try to develop a processor which is very fast. It will work on say some 200 mega Hertz clock or 500 mega Hertz clock or 1 gigahertz clock, right.

So, that means, when you try to develop certain such processor, your objective is to optimize the speed, try to improve the performance in terms of speed. Now, I told you to develop another processor with the same specification target for say power. I want to make it low powers processor from the same specification and now, I told you to develop another circuit or another processor which is less area, right.

So, I do not bother about the power or the speed. I want to, I am bother about the area, the size of the chip. I want to make a tiny processor for that, but it will be from the same specification again. So, you understand that from a same specification you might have various design optimization criteria and you have to develop different different processor for the same thing.

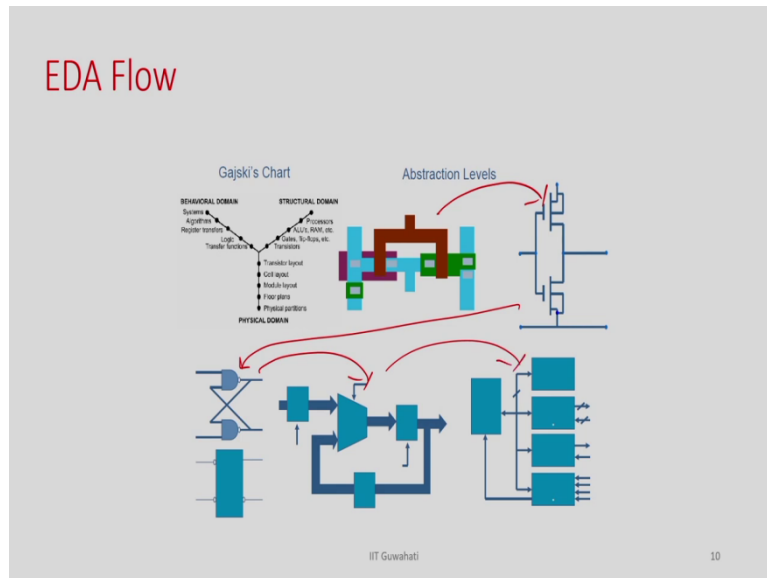
But, what about in RTL? You have to develop things from scratch, right. You have to develop a power efficient processor, you have to develop a area efficient processor, you have to develop a timing efficient processor or the high speed processor, right. So, that is what is called design space exploration. So, doing this in RTL, it is actually doubling or it is the complete same effort for all processors. It is a huge time, right. Something it is actually very difficult.

Whereas, if you start from a C code and say in the high level synthesis, you have certain parameters. You just specify that parameters in the tool that you do something for speed, you do something for area, you do something for power and the tool automatically give you these different different target processor in the RTL level within say minutes. So, that is a huge huge benefit, right.

So, this design space exploration is actually possible only with this C based VLSI design or if we start the things with at the high level behavior not at the RTL level. So, that is again, it is the biggest advantage. The other advantage is like fewer errors. Obviously, if your size is, code is small, the chances of having manual error, is something human error, is less.

And obviously, once you do in the higher abstraction level your verification effort will be less and then also you can actually apply different kinds of optimization, more optimization, at the higher abstraction level, right. So, these are the kind of advantages we will achieve through this design automation.

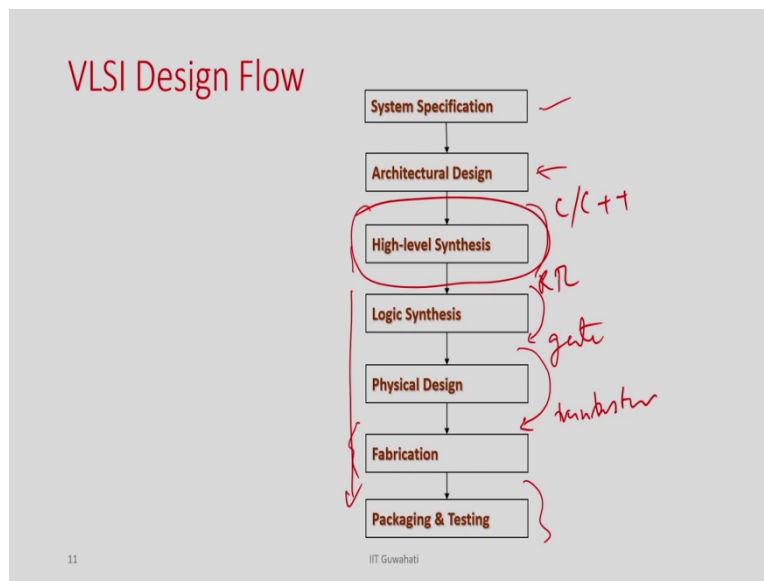
(Refer Slide Time: 17:03)



So, I have, as I already discussed that this electronic design automation flow is going through like this. Physically you have this layout. So, the layout has this, I mean physical things like this metal oxides and all those things. And then there are multiple layers and which is actually when you do this physically, you put these layers one after another and based on the layout information. It will actually perform this transistor behaviors, right.

So, actually this layout, this what we visualize this as the transistors and then from transistor we have the gate level thing, from gate level we have the RTL and from RTL, we have the specification or the high level C. This is one of the this EDA flow.

(Refer Slide Time: 17:50)



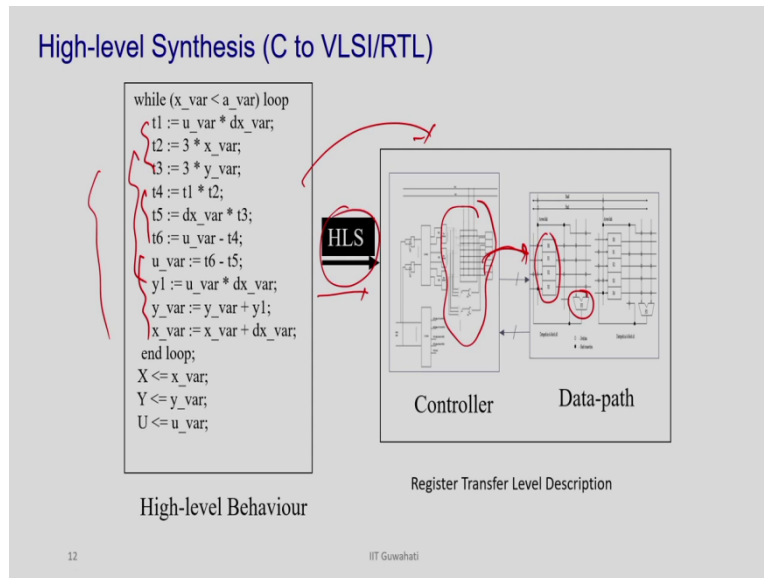
So, here I just summarize this VLSI flow or the EDA flow like this. So, you have the specification and then you write the architectural design or you can write a high level design and then you do the high level synthesis. You will get the RTL, right. So, this is your C, C plus plus, Python and then from this you will get gate level design using logic synthesis and then you do this physical design.

This physical design will give you the transistor level transistor level design and then you actually do the fabrication, right. So, fabrication is something where you actually physically put this metal oxide layers one by one which actually be based on the layouts and that will give you the physical IC, right.

So, this all is a software. They will just logically convert this, but physically you have to get the chip and where this fabrication do this. Once the fabrication of the IC is done then you just do the packaging and you just do the testing and make sure your IC is actually functionally correct and then you ship it to the market, ok. So, this is what the overall VLSI design flow.

So, I am going to cover these things very briefly just to tell you what is the overall flow. But I am going to cover more on this particular topic, right, the high level synthesis, in this particular course.

(Refer Slide Time: 19:07)



So, as I mentioned, this high level synthesis or the C based VLSI design, is something converting a high level behavior into a RTL design. So, in this RTL design, you can think about C code which has basically a loops, if else, anything, function call. So many things can be there in the in the C code and this tool will automatically convert into a RTL. And, the RTL, that is generated by a high level synthesis tool has a very specific structure. We will go into detail of that.

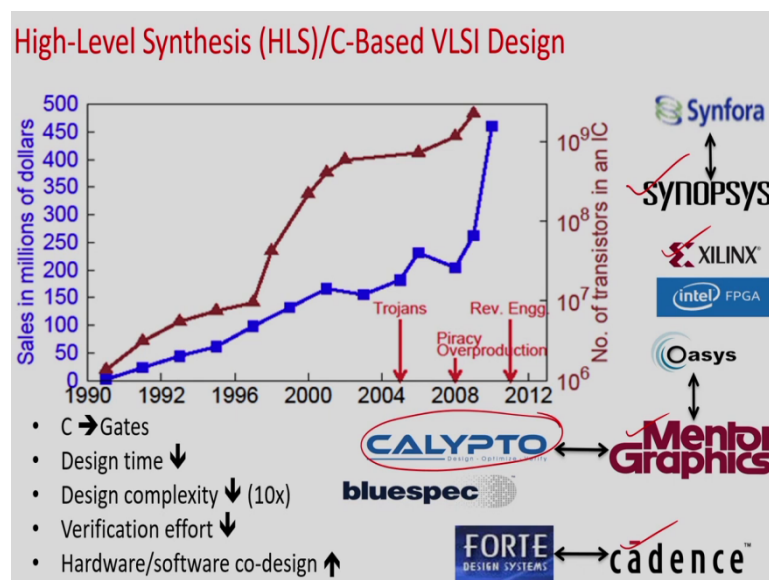
It consists of a data path, which actually execute these operations in the hardware. So, this data path consists of registers, memories, function units and their interconnections and there is a controller. So, controller determines that which operation to be executed in which clock, right. So, because this operations are executing. So, there are many operations. You are not able to execute everything in a single clock, right. So, it is not possible.

So, you have to basically decide that, in this first, in the clock 1, I am going to say do this three operation, in the next clock I am going to do these three and then next clock and so on. In the next clock, you are going to do this four operations and so on. Then this controller will say that, it will send the control signals to this data path to execute these first three operation in the data path in clock 1, in clock 2 the next thing and so on.

And, also whatever the control flow this while loop, so that these things will repeat for say 10 iterations and so on. If there is if else branch and all so, that will be controlled by this controller FSM. It is a finite state machine. So, then overall this combination of this controller and the data path is the register transfer level description and which will actually execute the same behavior that is specified by a high level C code, ok.

So, this is something is the conversion and in this course I am going to talk about the whole algorithm that is written inside this high level synthesis, how it actually works like this and all other detail I am going to cover in a few slides.

(Refer Slide Time: 21:17)



So, because of this huge advantage, bringing the high level synthesis bring to this EDA design or electronic design ah, electronic industry so, it is getting hugely popular day by day. So, you can see here almost all big houses, big EDA industries like Xilinx, Synopsys, Mentor Graphics, Cadence, they have a high level synthesis tool which is a commercial tool now.

So, everybody has their own own high level synthesis tool and they are getting popular and they are getting widely used in the industry and we are getting commercial chip that is developed through this high level synthesis tool. It is getting used to develop the practical ICs in current EDA industries, ok.

(Refer Slide Time: 22:06)

Specialized Hardware

- Why do need Specialized hardware?
- Why are the general purpose processors not sufficient?

14

So, obviously, I am able to make sense that this particular course is very important in the sense that we can actually use this particular C based VLSI design or a high level synthesis as the gateway to come to this hardware from C code.

So, somebody who is not expert in hardware concepts or a person like me who is a computer scientist who can actually take a C code and automatically convert this into an hardware or a RTL, which is actually then further synthesizing into logic synthesis, physical synthesis and I can actually develop IC from my code that is written in C.

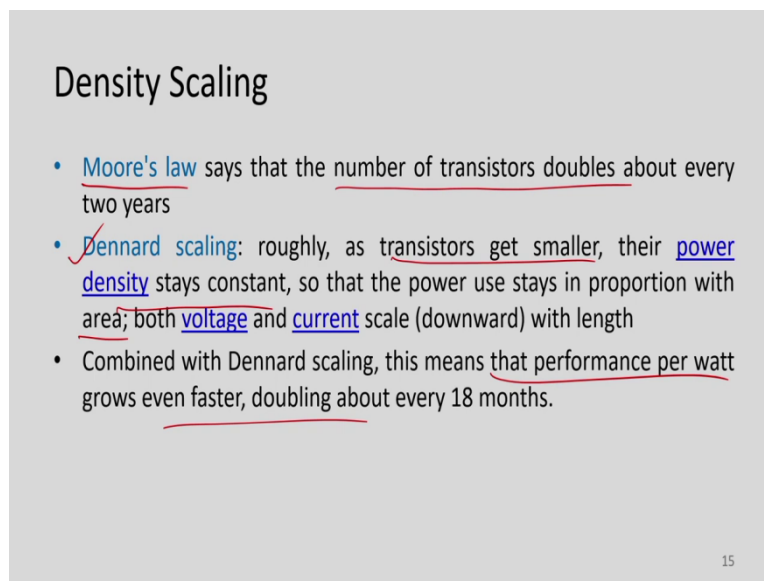
So, obviously, this looks exciting, but there is one more very big advantage that this particular high level synthesis brings in the current context and I am going to emphasis that in a few lectures, few slides. So, this is the era of specialized hardware, right. So, you have, you might be heard of all this IP intellectual property- one is for the jpeg encoder, say one something is for E-mail, one is for say a noise reducer, one is for say USB, one maybe with some some Bluetooth. So, for everything we have now a specialized hardware.

But, the question that you should ask, why we need specialized hardware, right. So, why we need some specialized software because we have general purpose processor and they are really powerful. If you think about a Intel processor which might have say 16 cores, 20 cores

or 32 cores; that means, the those number of processors and each of them is highly powerful. Why I cannot do this image processing thing inside this general purpose processor?

Why I cannot do this image reductions or say noise eliminations, all those operations using a general purpose processor? Why is the current trend is developing specialized IP for every purposes? Right. So, let us try to understand that.

(Refer Slide Time: 23:54)



Density Scaling

- Moore's law says that the number of transistors doubles about every two years
- Dennard scaling: roughly, as transistors get smaller, their power density stays constant, so that the power use stays in proportion with area; both voltage and current scale (downward) with length
- Combined with Dennard scaling, this means that performance per watt grows even faster, doubling about every 18 months.

15

So, for that, let us try to understand this overall the evolution again, this EDA industry. If you just know this Moore's law, which says the number of transistors doubles in every 2 years.

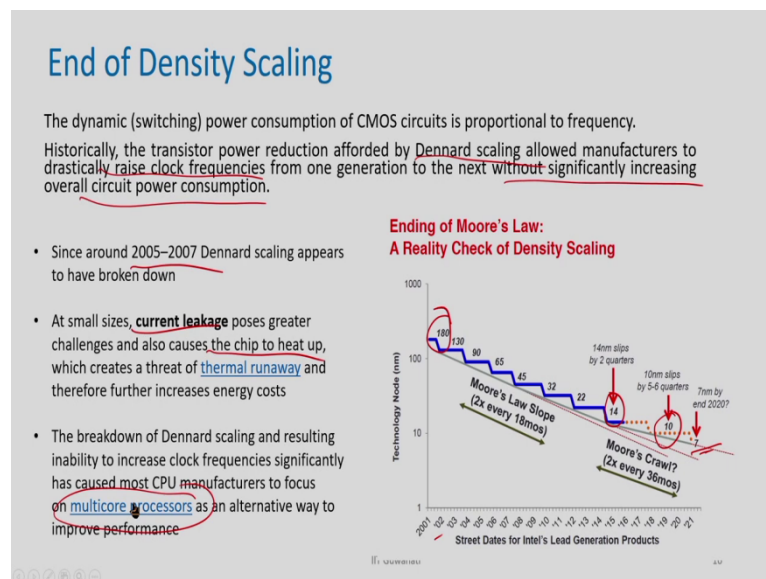
So, you understand that every day you try to do more operation. So, you need more resource, right. So, you need more transistors. So, Gordon Moore's, he observes that the the requirement is like this that the number of transistors inside a chip is getting double in every two years, roughly, right.

So, now the question here is that, if you have more transistors in a chip so, your size should grow. But, it is not happening. It is happening in other direction, right. So, it is actually day by day we are getting a tiny chip, right. We have a mobile which is very just like this, and we can actually keep it in the pocket and which has billion transistors, right, which is running in some gigahertz speed.

So, that means, the size is shrinking and the transistor is getting increased, right. So, how it impacted. So, that is something characterized by this Dennard scaling which says that since the transistor getting smaller, so, we can actually put more transistor in the same area. So, the power density does not increase, it stay constant. So it is proportional to the area. So, as long as your area remains same, you put more transistors in it, since it is a smaller size, the overall power density does not increase that much, right.

So, what does it implies? So, if we just combine this Moore's law with Dennard scaling, it means that you actually getting performance per watt is improving day by day, right. Because, you have more transistor, you have the same power, but you are doing more operation. So, the performance or the operation per watt is getting faster which is really good, right. So, is it something, it will continue? The question is will it continue forever? No, right.

(Refer Slide Time: 25:51)



Because if you see this particular diagram, in around 2002 we have a 180 nanometer transistors and now we have 14 nanometer, we have 10 nanometer nowadays. And we can go till 7 nanometers. So, that means, your transistor is getting smaller and smaller, right.

So, this say 10 nanometer is the distance between the gate and drain, that source and drain, right. So, which means your transistor getting very small, it very tiny day by day. Can it becomes 0? So, you have end of it, right. So, you cannot make your transistor after certain

time it is not possible to, I mean make it smaller after a certain period because which is not physically possible, right.

(Refer Slide Time: 26:46)

Power-Constrained Modern Computers

$\ll 1\text{W}/\text{chip}$ $\sim 1\text{W}/\text{chip}$ $\sim 15\text{W}/\text{chip}$ $\sim 50\text{W}/\text{chip}$ $\sim 100\text{W}/\text{chip}$ $>100\text{W}/\text{chip}$

$$Power = \frac{Energy}{Second} = \frac{Energy}{Op} \times \frac{Ops}{Second}$$

- ▶ **Energy efficiency must improve!**
- ▶ **Limitations of general-purpose multicore scaling**
 - Amdahl's law
 - Dark silicon

IIT Guwahati 17

So, but one more important aspect that since you want to keep your power consumption fixed for a circuit or you will try to reduce it since your energy per operation, so, power is basically energy per second and this energy per operation is getting reduced as I mentioned this is the overall performance is getting improved. So, you need to run it faster, right. So, you want to do more operations per second.

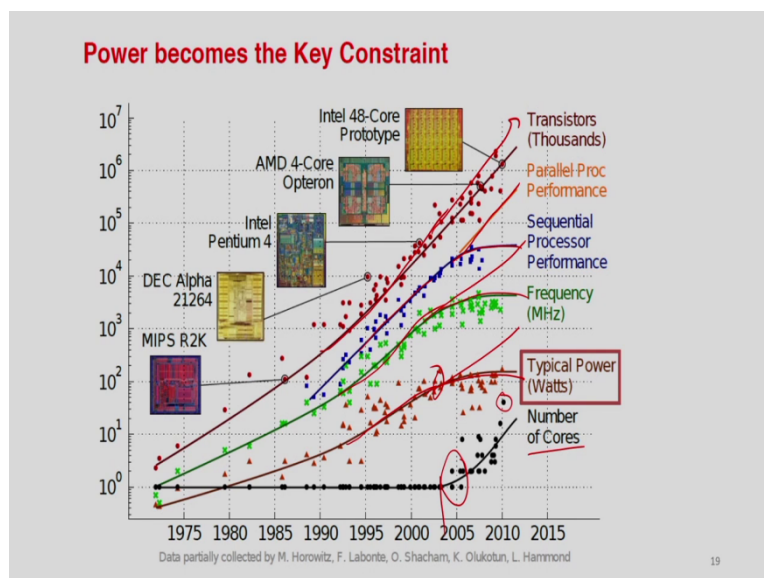
So, your frequency has to increase and that is the current trend that is to keep this scaling. So, we basically try to achieve the same power by significantly increasing. So, we try to drastically raise the clock frequency to keep the power consumption more or less same or not improved that much, right. So, that means, we try to put more transistors into a circuit and then you try to keep the power consumption same. So, basically you are increasing the frequency, but it has side effects, right.

So, basically once you have this transistor become tiny, there is a current leakage and also problem of heat up, right. So, your chip can burn with this many transistor working on that. So, around 2006 or 2007 we found the end of this density scaling. We cannot just reduce

these things or you cannot pack more transistors into a particular design, I mean as much as possible, because you have the side effect of current leakage and this heat up, right.

So, then we actually move to the era of multi core design, right so, multi core processors. So, you know that, instead of packing more transistors, you make it more complex let us distribute it, right. So, instead of using one very powerful transistor, let us try to put more processors. You have used multiple core or multiple processors. All of them may not be that powerful, little bit lower powerful and then you try to parallelize the whole thing, right. So, that is what this multi core processor, multi core era comes.

(Refer Slide Time: 28:36)



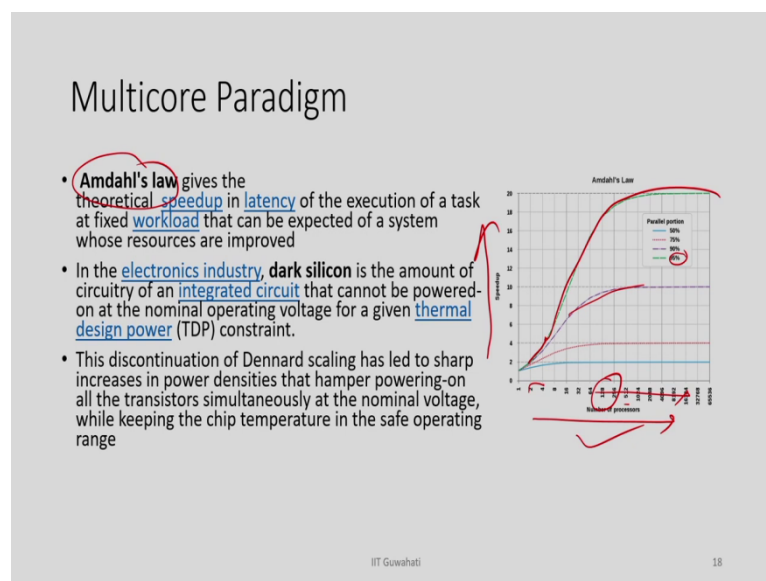
So, you can see this diagram perfectly fine. So, you see till 2005 or so, we have the single processor as the number of transistor grows. So, this is the number of transistor grows. So, as the number of transistors is growing almost exponentially, your overall power also getting increase and then the frequency also. Because, you have to increase the frequency and the performance also getting increased with the cost of higher frequency and higher power.

So, by that time it reaches almost 100, almost 100 watt and then it is basically, physically, you know difficult to place more than. Then we go to the era of multi core. So, this is a number of core instead of a single processor. I am actually putting a multiple cores and see, there are now, almost we have a 64 cores in a particular circuit design.

With that I am able to control the power consumption, because, it is not going like this. I am able to control the frequency. I can actually down the frequency and also the sequential processor performance does not improve. Because, I do not want that, but the overall performance using the parallel processor increase, right.

And, I can keep this Moore's law live because I am keep increasing the number of transistors in my design. So, is it something we will continue that I will put a 32, 64, 120 and so on. There is end of it that also, alright.

(Refer Slide Time: 29:57)



So, this is given by this law. This particular diagram told that how the speed up will happen with the number of processors. This is the number of processors, this is the speed up achieved. So, if you just double your processor so, your speed up might be doubled initially, but after some time it will flatten down, right. So, this is for 95 percent parallel and this is say for 50 percent. So, after certain time even if you increase the processor, you would not get that much of speed up, right.

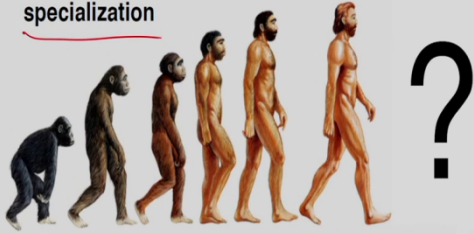
So, that means, so, after certain point, because it has now other overhead, right. So, that communications, synchronization, so many other overheads comes. So, eventually, you would not get those particular advantage or the performance benefit. So, then is the roadblock! That I mean initially I try to make a more complex chip and that has a end and then we try to make more more processor into a single chip.

And, now I am seeing that even if you put say more than 128 or something you cannot achieve. I mean does not matter how many processor you pack into single system on chip, you would not get much performance, right. So, is it a roadblock? I mean is the end of this? No, right?

(Refer Slide Time: 31:12)

Advance of Civilization

- ▶ For humans, Moore's Law scaling of the brain has ended a long time ago
 - Number of neurons and their firing rate did not change significantly
- ▶ Remarkable advancement of civilization via **specialization**



IIT Guwahati 20

So, what is the way out? Right. So, we are almost kind of a roadblock. What is the way out? So, if you look into this human civilization, advance of civilization so, as the day progressed, we try to have a specialization, right. So, basically the civilization advancement happened through specialization, right.

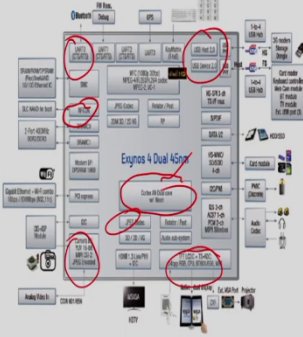
So, it means so, we have this doctor, we have engineers. Within the doctor we have various specialists of doctors and then in engineering also we have specialized of engineers, right?

Somebody is electrical, somebody is computer science, somebody is mechanical, civil and so on. So, and then this specialization actually makes this advancement things happen, right.

(Refer Slide Time: 31:52)

Computers are Following the Same Path

System on chip (SoC)



- ▶ Modern SoCs integrate a rich set of **specialized accelerators**
 - Speed up critical tasks
 - Reduce power consumption and cost
 - Increase energy efficiency

[source: Samsung]

IIT Guwahati 21

So, same thing actually, these computers also follow the same path, right. So, it means instead of try to do everything using a general purpose processor, why not we try to think about a specialized processor which will do a specific job. So, it will be tiny because it is doing only one job. It is not the single processor doing everything.

So, it will be a single specialized processor. So, it will actually do a specialized jobs. So, it must be very fast. Because, it is actually doing only that job. It is actually developed for doing one job.

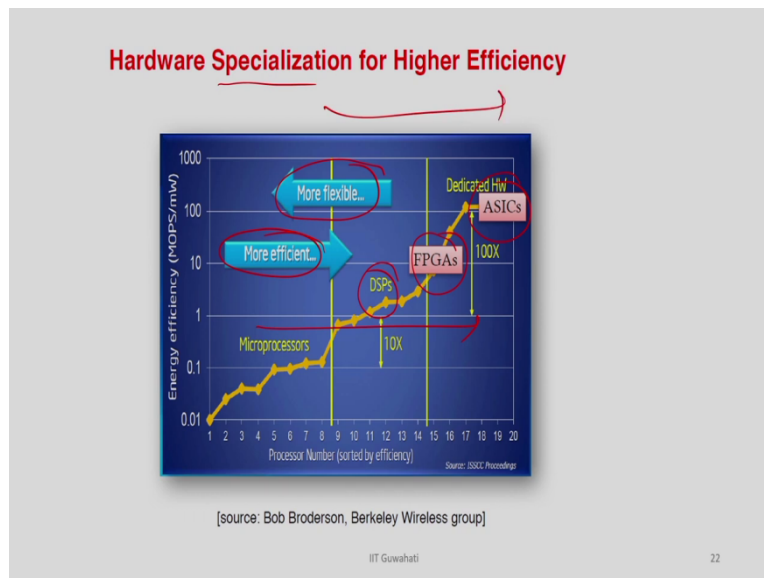
It will speed up the critical tasks because I know how to take care of certain difficulty of a particular job in that particular chip. It will reduce the power because it is doing only one thing. So, it has very small, it is not comparable with the general purpose processor and also it will increase the energy efficiency.

So, this is how this overall this evolution of this electronics design is coming and if you see this some Samsung chip, Samsung mobile that we are using in modern days. It has, there is a dual core processor, which is kind of working as a manager.

You have so many IPs, right, so many IPs. One is for say camera, one is for say jpeg, one is for say your interface, you have for USBs, well know so many, right. So, it has many such IPs or intellectual properties or a specialized hardware that is actually fit into a system on chip.

So, they are connected and they are just placed there to do a particular job, do quickly or for accurately and this particular processor is actually managing all this just coordinating with them and make sure that your whole chip is working fine, right. So, this is where the specialized hardware is something become extremely popular in modern days because to achieve the targets, right.

(Refer Slide Time: 33:43)



And, this specialized hardware has advantages and disadvantages. So, whenever you have a general purpose processor, it is more flexible, right. So, you can actually have a processor which can do image processing, it can do some AI things, it can do anything, right. It can actually do some video coding decoding and whatever it is. It can do anything right as long as you can map these things into the instruction set. So, it is more flexible, but it is less efficient, right.

But, when you go for the specialized hardware it is more efficient right. So, in the specialized hardware domain we have the DSP and then we have FPGA and then ASIC is basically is a

specialized tiny ICs for doing things, a particular job, right. So, that means, if you have an ASIC chip, which will do only one thing, right. In the FPGA we can map, it is a reconfigurable architecture. So, I can map different different design into this and I can actually execute that part.

So, with this the advantage and disadvantage, but we do not need the flexibility, we need the efficiency in the current context. So, this specialized hardware is something is the trained in the current electronic design automation.

(Refer Slide Time: 34:52)

Best of Times for Specialized Computing

Blue Chips	Startups	Academia
✓ Apple Google Intel Microsoft ...	Cambricon Cerebras Deephi (→ Xilinx) Graphcore ...	EIE/ESE [Han ISCA'16, FPGA'17] EyeRIS [Chen ISCA'16] FINN [Umuroglu FPGA'17] Minerva [Reagan ISCA'16] ...

✓ Deep Learning is causing a revolution AI and computer hardware industry

IT Guwahati 23

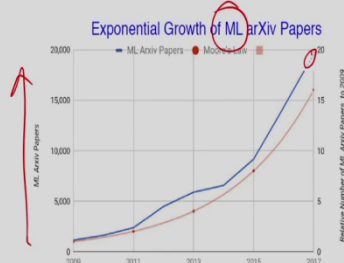
And, you can see that in the academic, in the startups, big companies, everybody try to develop a specialized chip, right. So, they try to develop a specialized chip for every purposes and specifically this deep learning and this revolution of AI make this industry more useful. Because, you need more specialized architecture, specialized hardware, specialized processor for all these deep learning algorithms, right.

(Refer Slide Time: 35:22)

“Worst of Times” Also?

Is RTL design method still viable?

- ▶ Target of specialization is constantly & rapidly moving!
 - In particular, machine learning (ML) algorithms and software frameworks are rapidly evolving



Jeff Dean, "Machine Learning for Systems and Systems for Machine Learning", 2017

IIT Guwahati

24

So, is it something that time also for these things because you see there is a funny picture here is that the number of paper published in this ML domain, right. So, this is archive paper only. So, we can see here almost 20000 paper getting archived every year in the ML domain. So, is not it the too many applications and if we try to develop this IC for all these 20000s, I mean is it not something too much if you try to develop them, right.

(Refer Slide Time: 35:51)

Development Effort of Hardware Accelerators

- ▶ Comparative study on Monte Carlo option pricing:
 - Data for a single option pricing, using 524,288 simulation paths

Platform	Normalized Speed-Up	Normalized Performance/Watt	Development Time in Days
FPGA	545:1	1090:1	60
GPU	50:1	21:1	3
GPP	1:1	1:1	1

Source: "Reconfigurable Computing in the Multi-Core Era," Khaled Benkrir, HEART'2010.

Conventional hardware design practice requires extensive hand-coding in RTL and manual tuning

25

So, let us try to understand this from this table that once you have a general purpose processor, if you just consider that is a speed up is 1. You go for GPU. It is basically 50x

improvement. In FPGA it is almost 550 times improvement, right. So, this is performance per watt also increase this way, right. So, it is almost 1000, I mean 1100 times faster than the normal processor in the performance, in terms of the power.

And, also the design time in increase because you can map anything into a general purpose processor, but here you have to develop the whole circuit. So, this is the kind of advantage where you have to look into, right and where our this course comes into handy ok. So, now, the question is that we have this many application, we need specialized chip for all these applications.

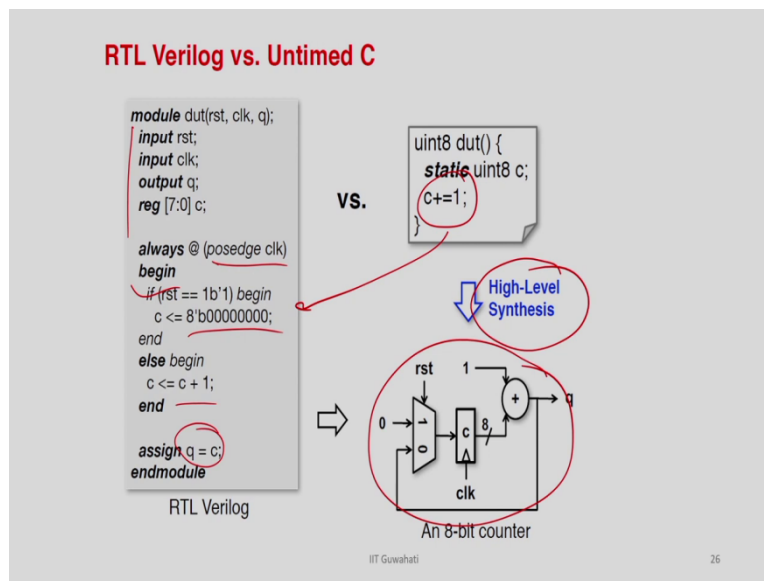
So, is it possible to develop everything in RTL? Right. So, there is the transfer level as I mentioned still it is a hardware design. It has clock, it has multiple clocks, it has multiple sync signals like synchronous signal, set, resets, it has memory, it has all this interconnection. You have to take care everything at this level. So, developing so many applications or the so many IPs at the RTL level may be difficult right if it is not impossible.

So, there this course, this high level synthesis course comes into picture, right. So, instead of developing IP at the register transfer level because all the application people write AI application never in RTL, right? They always write in say tensor flow, in C, C plus plus, right and then try to run it in a general purpose processor. What about this high level synthesis that will take those high level behavior in tensor flow or say C, C++ and then you convert them into equivalent hardware. That will do a wonder for this.

And, we can actually quickly get IP which is specifically doing a specific job right and also as I mentioned that we can actually have different, different IP for different purposes. One is for say speed, power and area, right. So, that is where this particular course is so relevant in the current context that we can actually take all this new developed ML and AI aspects

And then all this algorithm we can quickly take them and convert them into a specialized IP, which I can, or the specialized IC which I can actually utilize for faster execution of those behaviors, right. So, that is where this high level synthesis things comes into picture.

(Refer Slide Time: 38:20)



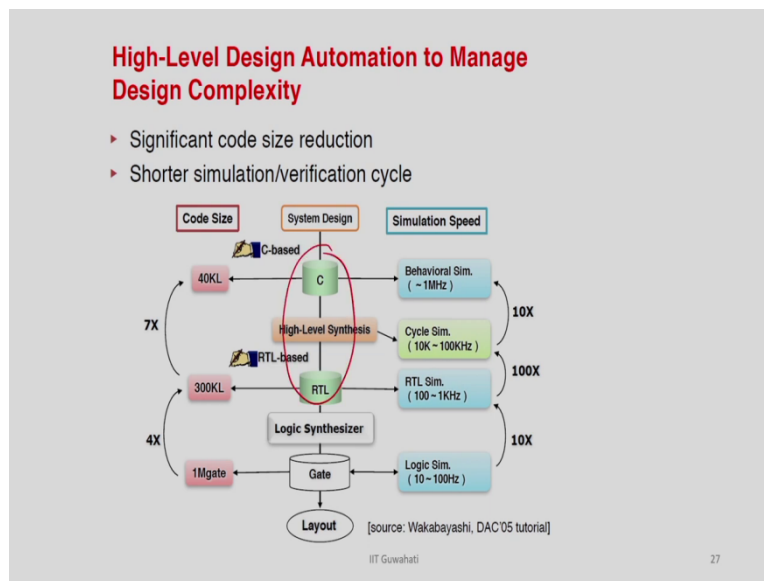
So, here I just given a very high example that you want to just develop a counter. So, it is just a C++, right, C equal to C plus 1 in C and if you just try to write this in hardware, you have to have a clock. In after every process of clock, you have to, if the reset comes you have to reset the things otherwise you just increase it and then you assign the output to this C, right.

And you have to develop this registers and all those things, right. So, this is how we have to write the RTL and whereas, this is nothing, but C equal to C plus 1 in the C, right. So, and then this high level synthesis tool take this and it will convert into this and which is nothing, but this circuit and then this is the high level synthesis tool, right.

So, this is where this particular course is so important because we need now specialized architecture, we need a hardware acceleration and for this hardware acceleration because almost everybody write everything in the higher level in specification in C, C plus plus, tensor flow.

And, then this high level synthesis can convert those applications into equivalent hardware which will be a very faster or efficient execution of this particular task in a hardware, right and which will give you more performance benefit over general purpose processor, ok.

(Refer Slide Time: 39:33)



So, this is where this particular course is so important and I am very excited that I am going to cover many important things in this course and this particular diagram again summarize that the use of this this or design automation and specifically the C to RTL, right. So, this is something I have already covered.

(Refer Slide Time: 39:53)

ML and EDA

- Use machine learning algorithms to improve the synthesis steps
 - Million of gates
 - Most of the synthesis problems are NP complete
 - Synthesis steps are heuristic based.
 - May not work well with new scenarios
 - I am Skeptical about the impact of ML in EDA.
- Use EDA to improve execution of machine learning algorithms
 - Use EDA tools to develop AI/ML specific processor
 - Arm Machine Learning Processor
 - A **tensor processing unit (TPU)** is an [AI accelerator application-specific integrated circuit \(ASIC\)](#) developed by [Google](#).
 - Automatic synthesis of ML algorithms to FPGA
 - FPGA implementations of ML applications can often run much faster than software implementations and can consume significantly less power.
 - Quite promising area of research.

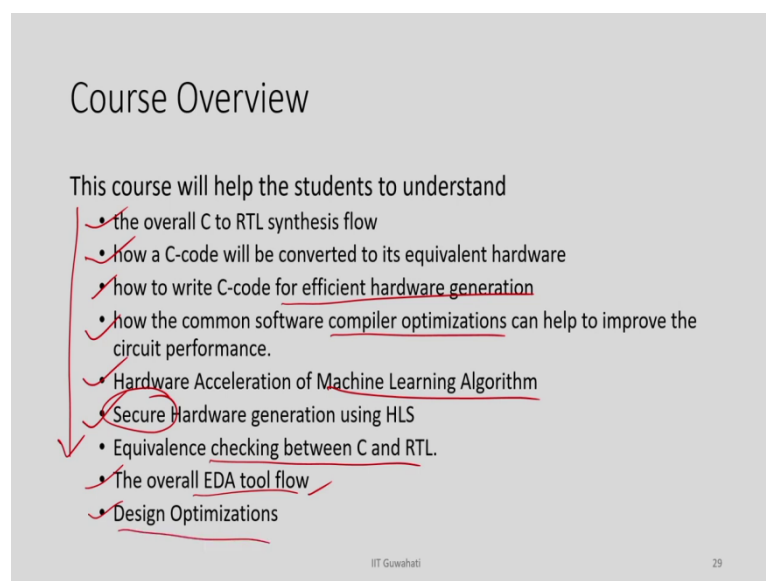
So, in the context of machine learning and electronic design automation there are two ways of use it. So, the first use is something like that we have the EDA algorithms. I told you that I

have a C code and then you convert into RTL using high level synthesis tool. I have say RTL code I want to convert into gate level design using logic synthesis tool. Can this machine learning tool help, machine learning algorithm helps there, right! So, that is one way of looking into application of ML in this EDA.

Similarly, you have one more direction is that which is more relevant for this course is that can this EDA helps to have a performance improvement of the machine learning algorithms, right! That is, can you have a hardware accelerator for machine learnings using this EDA tools.

And, if you look into this current context, there are a wide variety of such applications are coming and it is a very popular area of research. And in this particular area of research this high level synthesis is more useful and very relevant in this current context right.

(Refer Slide Time: 40:54)



The slide titled "Course Overview" lists the following topics:

- the overall C to RTL synthesis flow
- how a C-code will be converted to its equivalent hardware
- how to write C-code for efficient hardware generation
- how the common software compiler optimizations can help to improve the circuit performance.
- Hardware Acceleration of Machine Learning Algorithm
- Secure Hardware generation using HLS
- Equivalence checking between C and RTL.
- The overall EDA tool flow
- Design Optimizations

IIT Guwahati 29

So, now, I hope, I can convince you that this particular course is something very exciting and it is very important to develop a hardware accelerator for the modern day application like machine learning and deep learning and any other application as well. So, I am going to taught this particular course with keeping into mind in the current research in context and specifically I am going to cover, these are the topics in this high level synthesis.

The first thing we try to understand, how this C code is converted into RTL? How we convert untime code becomes a timed code, how to do all those things right. And then what are the steps involved here, right. So, what are the sub steps of high level synthesis and how I can logically convert these things step by step into an RTL. So, I am going to cover these things in detail. All the algorithm involved here, what are the popular algorithm, what are the important algorithms, all the algorithms, I mean useful in this context I am going to cover here.

And, then there are one more very important topic is that if you write any code and you try to develop a hardware that may not result in a good hardware, right. So, inefficient hardware. Sometime you have to keep your mind, keep it in mind that and this code will be converted into hardware. So, you need to write this code which will be suitable for hardware, right. So, you have to write a hardware efficiency code. How to write that?

So, if you know that if you write this it is bad for hardware. So, I am not going to write this way. I am going to write the code in different way. So, that is something I am going to cover again that how to write C code for efficient hardware generation. Then I am going to cover the topic like this. Once you have a C code, you have a GCC compiler or LLVM compiler they have hundreds of optimizations.

So, they take a C code and then convert into another optimized C code which is applying various optimizations. I hope you are aware of those optimizations. Now, since I am converting this C code into hardware, why not taking the advantage of this optimization this is already implemented, right. They are widely available.

So, I am going to see what is the impact of this compiler optimizations in the hardware. Is it something good, is it something bad? If something is good I can take them automatically. Because, already they are there, right. I have a C code. I can just first apply this GCC to optimize it and then optimized version I am going to convert into hardware, right.

So, this is I am going to cover then how this common compiler optimization help to improve the circuit performance. I am going to cover that. I am also going to cover this hardware

acceleration for machine learning algorithm which is something very important in the current context. And also there is another important is this security, right.

So, now a days, the security is very important every context and hardware security is specifically is very important that say in the current context you know this finally, this fabrication happened by a third party. That means, you develop your chip, you develop the layout and you give the layout to somebody who do only the fabrication because fabrication is very costly. Not every electronic design company can afford that ok.

So, now the third party has access of your very intellectual property, right and they can actually check it and they can over over produce it. They can have a IP piracy, they can develop their own taking this or say more dangerous thing, they can actually add some Trojan into your circuit, malicious code into your circuit such that once you ship it, it will malfunction. It will do something right or they might leak certain secure data to them.

So, there is a huge chance of having all these things, right. So, to stop that, you have to develop a hardware even if it goes to third party, they are not able to understand this. Even if they actually get this layout, they cannot use it because of certain things. You lock the circuit, you make this secure, right or they are not able to do anything, I mean change on this, right.

So, that is where this hardware security is very important in the current context and I am going to cover that part also that how to develop a secure hardware through high level synthesis. That is also very important in the current context. And, then I am going to talk about this verification of this RTL. Because, once you convert a C code into RTL, I have to, you have to make sure they are functionally correct, right.

So, your RTL is exactly functionally correct with your input C code and that is where I am going to talk about the how to actually check that, the equivalence checking between this C and RTL. And, then also I am going to cover, since since it is a C-based VLSI design, it is not only the RTL and then how this RTL is convert into gate level, gate level code is converted into transistor level and so on.

So, I am going to cover the overall EDA flow, but again it is briefly, because the rest of the part is kind of very conventional. I am going to cover very briefly about this overall EDA

flow. And, also I am going to cover this. This is also important that in the RTL level there are many optimizations available. So, and also in inside this high level synthesis tool also there are many optimizations available.

So, what are the optimizations available and how to utilize, what is their impact in the generated hardware. So, I am going to cover this design optimization things in this course as well, ok.

(Refer Slide Time: 46:01)

Pre-requisites:

- Basic knowledge of digital design
 - Register, Clock, FSM, Gates, etc.
- Basic knowledge of Data structures and algorithms
 - Graph Algorithms
 - Search/sort
 - Shortest paths etc

IIT Guwahati 30

So, the prerequisite I am expecting that you have very basic knowledge of digital design. You should understand what is register clock, what is gates and all. Just having a basic knowledge is fine; I know, I am not expecting that you have a very depth knowledge about that. So, just having a basic knowledge, giving a very clock code you should understand oh this is this nothing else, right.

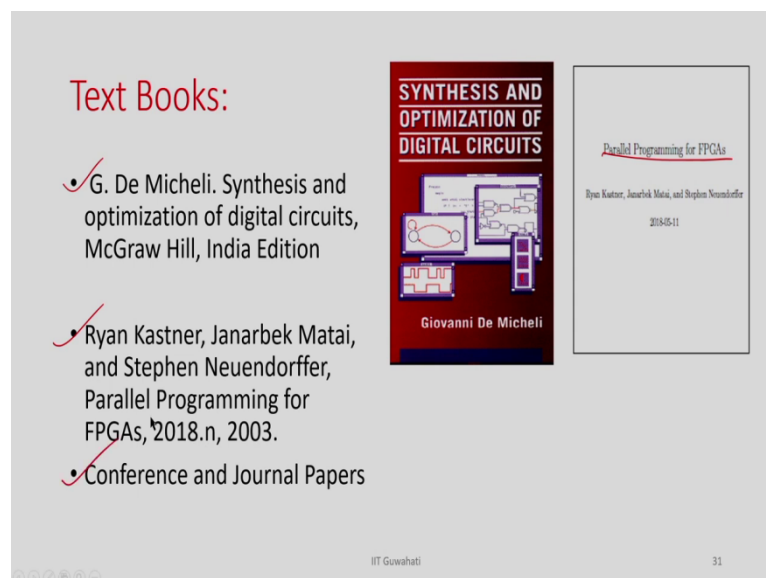
So, that is sufficient enough and also in this particular course, because, it is automation So, you need to be familiar with certain algorithm. Specially certain graph algorithms like graph coloring, click partitioning, shortest path and then sorting, searching and all those things, right.

So, you should have a basic knowledge about the data structure. That will help. Because sometime I want to take this say some scheduling problem or say particular optimization

problem into a say graph coloring problem. So, then I tried to solve using graph coloring right.

So, I am going to map this problem into very well-known computer algorithms and then try to solve the problem. So, having a basic knowledge about this data structure algorithm will help you to understand this course. But I will try to come make this course self-content. Wherever I need, I am going to give emphasis, a particular algorithm will explain, then I am going to use this. So, I mean, hope it do not create a big obstacle for you to understand the course.

(Refer Slide Time: 47:14)



Text Books:

- G. De Micheli. Synthesis and optimization of digital circuits, McGraw Hill, India Edition
- Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer, Parallel Programming for FPGAs, 2018.n, 2003.
- Conference and Journal Papers

IT Gowahati 31

Two textbooks I am going to follow in this course. Primarily this De Micheli's book on Synthesis and Optimization of Digital Circuits and this parallel programming of FPGA. This two book, I am going to follow and I am going to follow various conference and journal papers. Because, not everything is has a book on this. It is very advanced course and then it has many things which is already is still in research papers. So, I am going to cover many such important conference and research papers in this course.

(Refer Slide Time: 47:45)

Reference Books

- Steve Kilts, Advanced FPGA Design, Wiley, 2007.
- K. Parhi: VLSI Digital Signal Processing Systems: Design and Implementation, Jan 1999, Wiley.
- D. D. Gajski, N. D. Dutt, A.C.-H. Wu and S.Y.-L. Lin, High-Level Synthesis: Introduction to Chip and System Design, Springer, 1st edition, 1992
- Mike Fingeroff, High-Level Synthesis Blue Book, Mentor Graphics Corporation, 2010.

There are other reference books like this Advanced FPGA Design by Steve Kilts; Keshav Parhi for this various optimizations, Gajski book on High-Level Synthesis and this High-Level Synthesis Blue Book, right. So, these are the books I am going to and most of the books are actually I have learned. So, you can have it.

(Refer Slide Time: 48:00)

Acknowledgements

- These slides contain/adapt materials developed by
 - ✓ Prof. Jason Cong (UCLA)
 - ✓ Prof. Zhiru Zhang (Cornel)

Some of the slides I have taken this specifically these graphs, plots I mean, I have taken or adapted from these two professor. I am thankful to them for this content. So, with this I

conclude this lecture. I hope, you will learn a lot of things in this course and it will be a fun for you attending my subsequent lectures. Thank you.